Parallel Interactive Computing with PyTrilinos and IPython

Bill Spotz, SNL
(Brian Granger, Tech-X Corporation)
November 8, 2007
Trilinos Users Group Meeting
Albuquerque, NM
The History of Parallel PyTrilinos

• First versions of PyTrilinos were serial
• Marzio Sala implemented first parallel version
• Current status for MPI build:
  – from PyTrilinos import Epetra automatically calls MPI_Init()
  – MPI_Finalize() is registered with at_exit module
  – Epetra_MpiComm class is fully wrapped, and Epetra.PyComm() factory function is provided
  – $ mpirun -np 4 my_script.py works
  – A simple attempt at interactivity ($ mpirun -np 2 python) would produce multiple prompts and confuse standard input
• Attending tutorial sessions (SWIG)
• IPython tutorial included interactive parallelism demonstration (controller model)
• After I downloaded, compiled and installed required software, and IPython developers did the same with Trilinos, we got PyTrilinos to work interactively in parallel in about 5 minutes
IPython

- Enhanced version of python for interactive use
- Enhancements include:
  - Emacs/bash style tab-completion
  - Multiple layers of help (`help()`, `?`, `??`)
  - More context for exception tracebacks
  - Matlab-style input/output numbering w/cache
  - “Magic” commands, including shell commands
  - Generic shell access with the `!` prefix (`!!` If you want to capture output)
    - `$` prefix to expand python variables, `$$` prefix for environment variables
  - Logging
  - Much, much more …
IPython Demo
Parallelism with IPython

- Enable the rapid development of parallel codes
- Make all stages of parallel computing fully interactive: development, debugging, testing, execution, monitoring ...
- Make parallel computing collaborative
- Seamless integration with other tools: plotting, visualization, system shell, MPI, threads, etc
- Support many types of parallelism
IPython1 Architecture

Client → IPython Controller

IPython Controller → Instructions

IPython Controller ← Objects

IPython Engine

Objects

Instructions
IPython Engine

- Python interpreter connected to a network
- Runs user code, maintains state
- Can be started with mpirun
- User code can contain direct or indirect calls to MPI (such as PyTrilinos)
IPython Controller

- Provides asynchronous interface to a set of Engines
- Manages a queue for each Engine
- Clients push/pull commands and objects to/from the Engines using the queues
- A set of Engines can be presented to a client in a variety of ways. This enables different models of parallelism to be exposed.
IPython Clients

- Simple set of Python classes used to communicate with the Controller and Engines
- These classes can be used interactively or in non-interactive Python scripts
- Simple, high level interface
- Blocking and non-blocking modes
Startup Scripts

- **ipcontroller**
  - Used to start the Controller
  - The Controller listens on a number of ports and must be started first

- **ipengine**
  - Used to start a single Engine after the Controller is running
  - This script can be started using mpirun

- **ipcluster**
  - Starts one Controller and N Engines on localhost or an ssh based cluster.
IPython RemoteController Interface

- Simple, intuitive way of working with the IPython Engines
- Fine grained access to specific Engines
- Most general way of working with Engines
- New users should start here
- Designed with interactive usage of MPI applications in mind
If PyTrilinos and latest version of IPython1 are installed, it will “Just Work.”

The first time we attempted this, it took a few minutes to work out how and when `MPI_Init()` would be called.

We have added a command line option to `ipengine` to handle all of this automatically:

```
$ mpirun -n 4 ipengine --mpi=pytrilinos
```
IPython1 Parallel Demo
IPython1 Resources

- IPython Parallel Computing webpage
  - http://ipython.scipy.org/moin/Parallel_Computing
- IPython1 README and INSTALL files
- doc/examples directory in the IPython1 source

svn co http://ipython.scipy.org/svn/ipython/ipython1/trunk ipython
PyTrilinos for Trilinos Release 8.0

- **Extensive Teuchos.ParameterList support**
  - Python dictionary interoperability
- **Workable Teuchos::RCP support**
  - Python object are automatically ref-counted
  - Teuchos::RCP should be invisible to python programmer
  - Use cases where ref-counts do not sync
- **NOX re-enabled**
- **Module for Anasazi package added**
- **Python help system for PyTrilinos now leverages doxygen documentation**
• PyTrilinos requires that Trilinos libraries be built as shared

• Python-based build system that re-links object files as shared (supported under Mac OS X and Linux)
  – Must be compiled as position-independent code

• Shared libraries can be built without building PyTrilinos
  – configure ... --enable-shared ...
  – Shared libraries put in BUILD/packages/PyTrilinos/shared